How to / Maniaplanet / Manialink / Build a Manialink

Author : Marcel @ Funtrackers

LAST PAGE PAGE 1 2 3 4 5 6 7 8 9 NEXT PAGE

4. ManiaScript in ManiaLinks

ManiaScript is an independent language developed by Nadeo, to make different parts of the ManiaPlanet game more dynamic. This also applies to ManiaLinks: With ManiaPlanet, you have the possibility to manipulate and change the ManiaLink after it has been printed, and to react on the users behavior like mouse movement or keyboard inputs. Thus basically you can see ManiaScript the same way for ManiaLinks, as JavaScript is for HTML. This part of the Tutorial will exactly explain these ManiaScripts in ManiaLinks.

Hint: The Tutorial will not explain any syntax basics about ManiaScripts, so if you are not knowing anything about ManiaScripts yet, please refer to a ManiaScript Tutorial to get introduced into them. The following parts will require basic knowledge about ManiaScript.

4.1 Access to manialink elements

One of the most important things of ManiaScript in ManiaLinks is the manipulation of the elements of the ManiaLink. Before we can change any of the attributes of an element, we need to get our hands on this, i.e. we need to get the object instance, which represents the ManiaLink element. Basically, there are two different ways to get acces to an element of the ManiaLink, both together use the global variable Page:

Page.MainFrame.Controls[ElementID] – Access element over its parent frame
The first way of getting your hands on the instance of a ManiaLink element is to access it over its parent frame. The field MainFrame of type CGameManialinkFrame represents the <manialink> tag of the ManiaLink, and as the typename already says, it is simply seen as another Frame. A Frame now have a field Controls, which is an associative array mapping ControlIDs (of type Text) to the actual elements (of type CGameManialinkControl, see below for more information). The IDs are exactly those, which have been specified with the id attribute within the elements, so if you want to access an element, make sure it has an unique ID.

As Controls is an array, you also have the possibility to access its elements with their index. For example, Page.MainFrame.Controls[3] would give you the 4th child of the <manialink> (Remember that the first element has index 0), also if it has no named ID itself. Nevertheless, specifying IDs for your elements is the better way, as changing the order of the elements or adding another one may break any indizes you use to access these elements. So be careful with this.

Hint: In the Controls array, you only find the direct childs of the Frame, i.e. all direct childs of <manialink> in case of Page.MainFrame. This means if you have a Frame, you only will see this Frame itself in the Page.MainFrame.Controls array, but none of the elements contained in the Frame. If you want to access an inner element, you have to read the Frame (and cast it to CGameManialinkFrame), and then call this Frame's Controls array to finally get the wanted element. In the example below, this situation have been shown for better understanding.

Page.GetFirstChild(ElementID) – Searching for the element
The second way on accessing an element of the ManiaLink is the Page.GetFirstChild(ElementID) method call: You specify the ID of the Element you want, and the method will search for this element in the whole ManiaLinks. This means, if the wanted element have placed into Frames, this method will still find it.

Instead of searching all elements of the ManiaLink using Page.GetFirstChild(), you may, too, use this method on a Frame to only search the Frames direct and indirect childs for the specified ControlID. Especially Page.MainFrame.GetFirstChild() is aquivalent to Page.GetFirstChild(). After explaining the ways of accessing the elements, a short example should help to make everything clear, especially the issue with nested frames of the first way.

Example:
Example of accessing some elements of the ManiaLink:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<manialink version="1">
<timeout>0</timeout>
<quad id="exampleQuad" />
<frame id="outerFrame">
<label id="exampleLabel" />
<frame id="innerFrame">
<entry id="exampleEntry" />
</frame>
</frame>
<script><!--

declare CGameManialinkQuad quad;
declare CGameManialinkLabel label;
declare CGameManialinkEntry entry;
declare CGameManialinkFrame frame;

// Access elements over Controls Array
quad = (Page.MainFrame.Controls["exampleQuad"] as CGameManialinkQuad);

// We cannot access exampleLabel and exampleEntry directly over Controls,
// we have first to get the Frame, and call its Controls and so on.
frame = (Page.MainFrame.Controls["outerFrame"] as CGameManialinkFrame);
label = (frame.Controls["exampleLabel"] as CGameManialinkLabel);

// Now we set frame to the innerFrame element, to access the exampleEntry
frame = (frame.Controls["innerFrame"] as CGameManialinkFrame);
entry = (frame.Controls["exampleEntry"] as CGameManialinkEntry);

// The same elements accessed via GetFirstChild()
quad = (Page.GetFirstChild("exampleQuad") as CGameManialinkQuad);
label = (Page.GetFirstChild("exampleLabel") as CGameManialinkLabel);
entry = (Page.GetFirstChild("exampleEntry") as CGameManialinkEntry);
--></script>
</manialink>
```

So what is the advantage of the Controls array, if the method GetFirstChild() seems to be much simpler? It is the full availability of the element tree: You still see, that the innerFrame is inside the outerFrame, and exactly this can be helpful. Imagine, you have a Dropdown list, where you place all items of the list inside a separate frame. Instead of accessing all elements via an unique ID using GetFirstChild, you may simply loop with a foreach over all elements of the Controls array of the Frame to get all items of the list (and you may even omit IDs for these list items). This is not possible with GetFirstChild, as you don't have the element tree which says, that all elements in a particular frame are list items.

Another point of the example, which may confuse you, is the typecasting of the elements: Whenever an element is read, it is casted to its actual type. The reason for this is, that the Controls array and the GetFirstChild method always return an object of the class CGameManialinkControl. But as we may want the extended features of the elements, we have to cast them to their actual type. Only with casting the outerFrame to CGameManialinkFrame, we are able to use its Controls array to access the innerFrame. A detailed description of the available classes can be found below.

4.2 Manialink specific classes

For manipulating the elements of a ManiaLink, ManiaScript now provides different classes helping you. The base class, which is always returned by the Controls array and GetFirstChild(), is CGameManialinkControl. This class provides basic fields and methods to change the position and visibility of the element. All other classes are derived from this base class, and extend the possibilities. For example, CGameManialinkLabel has a method SetText() to update the displayed text. The following listing shows each of the available classes for the ManiaLink elements, which element they represent, and which class they are derived from.

CGameManialinkControl
Fields

- Text Id – The ID of the Control
- Real PosnX – The X position in the Menu coordinate system of the Control
- Real PosnY – The Y position in the Menu coordinate system of the Control
- Real PosnZ – The Z position in the Menu coordinate system of the Control

Methods

- Void Hide() – Hides the Control from the ManiaLink.
- Void Show() – Shows the Control on the ManiaLink if it has been hidden.
- Void Unload() – Completely removes the Control from the ManiaLink.
  The Control is not available any longer, and accessing the variable afterwards will result in an error.

CGameManialinkFrame
Extends: CGameManialinkControl
Represents: <frame>
Fields

- Array CGameManialinkControl[Text] Controls – The Controls this Frame contains

Methods

- CGameManialinkControl GetFirstChild(Text) – Provides access to child elements of the Frame
- Parameters: Text – The ControlID of the element

CGameManialinkQuad
Extends: CGameManialinkControl
Represents: <quad>

Fields

- Void ChangeImageUrl(Text) – Changes the Image of the Quad by specifying a new URL.
- Parameters: Text – The new URL to be shown in the Quad.
- **Hint**: The URL must be an absolute URL and the Quad must already have an image set with the image attribute to make this method work.

Methods

- Void ChangeImageUrl(Text) – Changes the Image of the Quad by specifying a new URL.
- Parameters: Text – The new URL to be shown in the Quad.
- **Hint**: The URL must be an absolute URL and the Quad must already have an image set with the image attribute to make this method work.

CGameManialinkLabel
Extends: CGameManialinkControl
Represents: <label>

Methods

- Void SetText(Text) – Updates the Text currently shown in the Label.
- Parameters: Text – The new Text to be displayed in the Label.

CGameManialinkEntry
Extends: CGameManialinkControl
Represents: <entry>

Fields

- Text Value – The current Value of the Entry

CGameManialinkFileEntry
Extends: CGameManialinkEntry
Represents: <fileentry>

Fields

- Text FullFileName [Read-only] – Provides the full path of the currently selected file of the FileEntry

Hint: Another tag, which does not have its own class, is <audio>. This tag is represented directly by CGameManialinkControl and therefore only has the basic manipulation of the position and visibility. The <manialink> tag is a CGameManialinkFrame and always accessable over Page.MainFrame. Not positionable tags like <music> or <format> are not accessable via ID at all, as changing e.g. their position would make no sense.

LAST PAGE PAGE 1 2 3 4 5 6 7 8 9 NEXT PAGE

This site is friendly to stationary (PC/MAC) and mobile (Android/iOS) browsers
Trackmania, Shootmania, Questmania games & media © Nadeo & UbiSoft used under fair use policy
8,373,037 page views since January 2008. All tutorials, services, coding © Mania Creative 2008-2013